AD-A055 254 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
A DISTRIBUTED MINI-COMPUTER NETWORK.(U)
DEC 77 R C ADAMS

UNCLASSIFIED AFIT/GCS/EE/77-4 NL

1 OF 1
AD
A055254

END
DATE
FILMED
7-78
DDC

AFIT/GCS/EE/77-4

14

1

6 A DISTRIBUTED MINI-COMPUTER NETWORK.

9 Master's THESIS,

GCS/EE/77-4

10 R. Cade/Adams
Capt USAF

11 Dec 77

12 93p.

This document has been approved for public release and sale; its distribution is unlimited.

012 225

AFIT/GCS/EE/77-4

A DISTRIBUTED MINI-COMPUTER NETWORK

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

R. Cade Adams, B.S.

Captain USAF

Graduate Computer Science

December 1977

Approved for public release; distribution unlimited

Contents

List of Figures

Abstract

This thesis presents a set of functional requirements for a distributed mini-computer network along with a basic architecture design to fulfill those requirements. The functional requirements were established by comparing the requirements of existing networks to the needs of the Air Force Institute of Technology Digital Engineering Laboratory. The requirements must provide a tool for education in operating systems studies while allowing mini-computer-based research.

The primary emphasis of the proposed architecture is the separation of the network operating system functions into independent prooesses which will run concurrently on the various mini-computers in the network. This distribution of network supervisory functions is made possible by limiting all communication between processes to a set of synchronizing messages. The messages provide a data interface which allows communication without knowledge of the processes location within the network.

This investigation also discusses a set of processes which will perform the functions set forth in the requirements. A brief explanation of the functional working of each necessary process is provided.

A DISTRIBUTED MINI-COMPUTER NETWORK

I. Introduction

The decade of the '70's has opened the computer revolution so that the computer is no longer a tool of only big business and big government. Today the computer affects all parts of human endeavor; including small business, even the smallest research efforts, and now even the average person's life at home. The obvious question then, is how have computers so infiltrated all aspects of our way of life. One answer lies in the evolution of the computer from large, expensive, removed mainframe number crunchers into the newest generation of compact, inexpensive, personally available mini and micro computers. These small computers make it possible to automate any task which can be logically thought out. Also, due to their low cost and availability, micro and mini computers are entering all research areas no matter how small.

AFIT is obviously a prime market for the low cost computing power offered by the mini-computer, and the Digital Engineering Laboratory is especially well suited for the use of small computers to allow both students and faculty the opportunity for personal research and educational time on the computer. AFIT, like many other universities, has found that these small "individual" computers are ideal for teaching students the basics of computer science by allowing an increased amount of hands-on work. It has been found that the principles are basically the same in the programming of the minis versus the larger mainframe computers. Also, the student can more readily understand the principles when he himself has the chance to interface with the system instead of going through an operator or operating system of great complexity.

Background

The previously stated reasons illustrate why the mini-computer is rapidly augmenting the large, mainframe computer at many research institutions. There are other factors, however, which keep the minis from completely displacing the mainframe computer from its research role. The mini-computer has limitations which must be overcome before it can act as a major aid to its big brother the mainframe computer.

The first and major limitation is the size of main memory of the average mini-computer. Most minis have a maximum of 64 K (kilowords) of main memory where the average mainframe computer may have a maximum of 2 million K of main memory. Obviously, with this capacity difference the mini is no match for the larger computers on memory size alone.

A second major limitation is the cost ratio of IO devices to the mini. A mainframe computer costs many times that of a single IO device; it is, therefore, easy to justify attaching many IO devices to the larger computers to facilitate it's use. The mini-computers present a completely different picture. One wishes to have a number of minis to facilitate their individual use by personnel, but to make them really useful each mini should have at least one batch form of input and output (card reader and line printer), some form of mass storage (tape drives or disk drives), and at least one interactive terminal (teletype or CRT terminal). An assemblage of these IO devices could easily cost two to five times the cost of the mini. This directly opposes the objective of the low cost individual computers.

An additional drawback to dedicated IO devices on the minis is the fact that they will surely have low useage. One of the prime reasons to have these research minis is to allow human intervention. Whenever a system interacts with people it must slow down to the human speed and, therefore, leave much time unused. Also, if these costly IO devices on one mini are not available to another mini, they cannot be used by a program running on the second mini even though they are not being used by the first mini.

A similar problem occurs with software preparation facilities (compilers, editors, linking programs). If one mini has all the appropriate software preparation facilities and someone wants to use a different mini that doesn't have them, then obviously they are being wasted on the former mini. One way of solving this problem is to have complete facilities on every mini, but as stated before this necessitates a large expenditure for capacity that has low usage, and this is not a good solution to the problem.

Problem Statement

The AFIT Electrical Engineering Digital Laboratory presently has six mini-computers, and is planning to acquire more. If properly outfitted, they make ideal tools for the various forms of research being conducted in the laboratories. Herein lies the problem with the AFIT minis. Even though the laboratory has six minis now, it has only a limited number of input output devices and fewer mass storage devices.

The problem is that only certain minis at any time can have the proper combination of batch IO devices plus interactive devices to allow much meaningful research to be carried out. Even worse, the available batch IO devices cannot be easily moved from one mini to another, so if one needs to do research on the minis without the batch IO, it becomes a long process of doing IO at the slow teletype rate (110 baud). While there are enough interactive terminals to allow each mini at least one device, the batch IO devices and mass storage devices are definitely outnumbered.

An additional problem is that of development software. Even on the best hardware equipped mini the development software (compilers, etc.) often make the user run a paper tape through the machine three times just to get an object tape punched out, which must then be loaded back in the computer in order to accomplish one test run. This procedure often takes in excess of 15 minutes. In this present age of interactive terminals and interpreters which allow instant compilation and execution of programs within one minute or less, this is truly a hindrance to the researcher trying to work on an ill-equipped mini.

Proposed Solution and Scope

The most encouraging part of the AFIT mini-computer problem is that it's answer lies within the laboratory itself, which already has most of the mini computing power and IO devices which are needed to allow a great amount of research to be done in an efficient manner. The solution lies in orchestrating the individual minis to act together and complement each other's devices instead of acting as small, distinct, separate entities which compete to keep devices solely to themselves. The lab even has access to the CDC 6600's vast development software provided through the INTERCOM interactive terminal system, if it were only channeled correctly so that the mini researcher could avail himself of its various compilation and editing facilities.

How can these minis, their IO devices, and the CDC's or other large scale computer's development software be combined? Through the use of a computer network. Numerous educational institutions have started to solve their mini usage problems in this way. By tying the individual minis together in a network, and linking that network to a large mainframe computer, they not only have the desirable aspects of hands-on work for the researchers at the mini-computer level, but they make the superior software development programs of the larger computer available at the same time.

This is the proposal of this thesis - the requirement definition of a mini-computer network which will interconnect all minis in the Digital Engineering Laboratory and provide an interface between this mini network and the CDC INTERCOM terminal system or other large scale computer system. This will allow any mini in the network to have access to any IO device in the network and to a large scale computer.

The scope of this thesis is not that of implementation, but that of a requirements definition and basic architecture design. The scope includes a study to produce the desired requirements needed to allow students and faculty to carry on research in an efficient manner of the proposed computer network, and a basic architecture design which will be used as a guideline for future implementation by other graduate thesis efforts.

Overview of the Thesis

The succeeding chapters of this thesis will describe an investigation into the requirements and basic architecture of a distributed mini-computer network. Chapter II will discuss the functional requirements deemed necessary to provide a complete network architecture which will meet the needs of the Digital Engineering Laboratory. Many existing networks were reviewed and the applicable requirements included in this chapter.

Chapter III outlines a basic hardware and software configuration which will fulfill the functional requirements. One software implementation scheme is presented which divides all network services into independent processes that run on the various network mini-computers, thus providing the distribution of the network operating system. This chapter gives the general structure of the operating system without emphasis on detail.

Chapter IV defines the nature of the system processes discussed above and shows how these processes can control their own communication and synchronization. Since the process is the building block of the proposed network, this chapter is necessary to present the basic structure and capabilities of the process unit.

Chapters V and VI describe specific processes which will implement the operating system functions and capabilities given in the functional requirements. Chapter V describes the basic functions needed to operate one processor in a stand alone capacity. Chapter VI describes the additional functions required to operate the full network of processors in a distributed computing capacity.

Chapter VII closes with the summary of conclusions and recommendations for follow-on efforts.

II. NETWORK FUNCTIONAL REQUIREMENTS

Introduction

Before a system of any type may be properly designed, the designer must have a firm idea of what the system will be required to do, and what limitations will be placed on the system while it is accomplishing these goals or requirements. This chapter will describe those requirements to be fulfilled by the proposed mini-computer network. Note that the requirements will be functional in nature. They will not try to assume how the functions are to be carried out; instead, they tell only which functions must be included in a proper design effort.

In compiling these requirements a number of present network requirements and implementations were reviewed. These requirements, together with the author's previous experience in working with small computer systems, were evaluated against the primary goals of the minis in the Digital Engineering Laboratory. A literature search was made of government research efforts into mini networks, and their various requirements were compared to the determined needs of the AFIT Laboratory. After conferring with faculty associated with the Digital Engineering Laboratory the following functional requirements were selected as the foundation for the proposed AFIT mini-computer network.

Uniform Processor Environment

The mini network will be composed of a minimum of four different processing units (processors). Since these minicomputers are made by separate manufacturers, they have slightly differing ways of performing their hardware functions. These differences are, in most respects, very minor and should be of no concern to either a network user or a network major sub-system software program (process). All functional software processes (a user's job or a network system function such as core allocation) should have a uniform processor environment so that the process notices no difference when running on the various processors.

The network must create a virtual machine at the lowest possible level. This means that the network will be seen by its users as a different machine than that defined by the actual hardware. The lowest level routines which handle the hardware itself should provide a standard interface to the rest of the network software so that both user programs and major network functions alike run on this standard, higher level, virtual machine. It is not sufficient to merely isolate the user processes from the idiosyncrasies of the different processors; the major system (network) functions (core allocation, scheduling, communication between processors) should also be isolated so that they may be written once and not customized for each processor. This allows more efficient maintenance and modification of network functions while maintaining a standard across the network.

By creating a virtual machine the network becomes conceptually one computer with multiple processors. A process can then communicate with any other process in the network regardless of which processor the processes are running on. This virtual machine environment must standardize formats for communication between processes and processors. The processes must see the entire network as one virtual processor with all processes running together. All communication between processes is then handled by the network in one standard format so that all processes will conform to the standard.

Dynamic Network Configuration

The Digital Engineering Laboratory serves many faculty members and students who are working on a variety of widely differing research efforts. Equipment is constantly being added to the Laboratory and much of it is being tied to the minis in some way. Hardware research projects are also being conducted which try one hardware configuration one day and a changed configuration the next. If the mini network is to be responsive to these research efforts, it must be capable of almost instantaneous reconfiguration.

The proposed network must also allow flexible assignment of IO devices to any of the processors in the network to provide for the constant influx of new devices and their reassignment from one processor to another in order to facilitate the equal use of devices among processors. The network must not be incapacitated by the loss of any one processor or IO device. It must be able to adjust its configuration and continue execution in a partial configuration upon recognition of a non-recoverable error condition.

All of the above constraints require that the network be able to dynamically reconfigure itself. This means that the network software must be able to handle these changes without reprogramming. Some form of status must be input to the network at start up time which the network will use to automatically configure its system software to handle the processors and their associated IO devices in their indicated configuration. Additionally, the network will periodically check this status to decide if the configuration of operational processors and IO devices has changed, and if a change is found be able to adjust the software to handle this change without disrupting unchanged parts of the network.

Symbolic Device Access

A process running on the network assumes it is running on a virtual machine; therefore, it also assumes that any IO device attached to any mini in the network is also a part of that machine. The network then has many terminals, paper tape readers, printers and other devices all accessible to any network process. By the previous dynamic network configuration requirement all IO devices must be interchangeable among the processors for configuration flexibility; hence, the process must be able to communicate with a selected device wherever it is located.

Symbolic addressing of the network IO devices provides this location flexibility. The network has information on its present configuration and can locate any desired device given some unique device symbol or class symbol. Class symbols would be used if the process wanted some class of devices (tape drive, disk) instead of a particular device (TAPE 1, DISK 2). One additional subclass should be provided - that of either LOCAL or GLOBAL classes of devices. Thus, the process can ask for a local device (one physically attached to the processor on which the process is running) or a global device (one anywhere in the network). This subclassing of devices is needed to facilitate the elimination of unwanted interprocessor traffic to global IO devices when local IO devices are available to the process.

With this identification of devices by unique device symbols, class symbols, and device subclasses (LOCAL or GLOBAL) the virtual machine (NETWORK) isolates the process from device location. It also provides for creation of virtual IO devices, allowing the network to use mass storage (disk storage) to simulate more devices than the network physically has.

Network Wide Data Base

The proposed mini network must have a flexible IO device configuration that allows IO relocation among the processors. This means certain files may also be moved from one processor to another (a disk file would obviously have to be transferred along with a disk drive if it were the only one on a particular processor). The network will then have movable files and should have a virtual or network wide base approach. The process again has no need to know where a file is physically located provided it can access it and insure certain class storage attributes.

A process must be able to choose between the type of storage devices used for a given file or that the storage device is of no concern. Again the local and global subclass attributes must be provided for confinement of highly active files to the local processor if desired. Provisions for copying and moving files from processor to processor by a single directive should be included. This requirement creates a higher level virtual machine by freeing the user process from processor dependent file locations. The philosophy is to free the process from knowing its environment by making the network processors, IO devices, and data base a single virtual element to the using process.

Modular-Separable Supervisory Functions

This mini network is proposed to be a virtual machine eliminating all possible processor dependency. This applies not only to the user process, but supervisory network processes too. The overall supervisory function must be divided into absolutely functional modules. The machines in this network are small in main memory size and must not be overloaded with unused supervisory functions; therefore, the functions of supervision must be carefully separated so that only those needed by a specific processor are provided.

Even this may not be enough. Commonly, supervisory functions are divided into modules, but the modules are highly dependent on each other (coupled) through complex status tables. This network requires that these modules be separable - all interactions between modules should be of a message type that may be queued by the receiving module so that modules are independent and could run concurrently if necessary.

This concurrency attribute will be used to allow one processor to contain supervisory processes that control other user processes on a <u>different</u> processor. It is expected that one processor may be dedicated to such supervisory processes as data base management and IO device configuration. If these functions were not separable from others, every processor would need its own copy of these commonly required supervisory functions. A second benefit of separable functions is the ease of modification. One network use is to be research of operating systems, and thus separable functions allow easy replacement of trial algorithms and experimental functional updates.

For the network to be a true virtual machine <u>all</u> processes must operate with a freedom from environmental restrictions. For the supervisory functions to obtain this freedom they must be not only modular but separable as well.

Error Recovery and Confinement

This network will be used as a research facility by both students and faculty. This research will not be limited to programs run on the network, but will extend into the systems and sub-systems inside the network software as experimental operating system methods are tried. Since the outside programs run on the network and the network sub-systems themselves will be subjected to constant experiment and changes, the network software and hardware will be highly susceptible to errors. Consequently, the network must be designed to try to automatically recover from these errors and if necessary confine unrecoverable errors to the smallest extent possible.

The network must be able to detect those errors which might cause networkwide failure. These errors may be in the user program or in the network itself. Here the necessity of separable supervisory functions is seen again. The system modules must protect themselves by monitoring each other's status and taking appropriate action when a failure is discovered. For example, the system modules which schedule the processes for execution time on each separate processor could communicate with each other and keep backup tables on the processes executing in remote processors. In this way the process status would be maintained in two separate processors. If one process scheduler failed, the backup scheduler would have the capability to try to reload the failing module from mass storage, reset the status, and try to resume normal operation. If the scheduler failed again, that processor might have to be disconnected from the network by the backup process. Accordingly, when any network sub-system fails, the remaining sub-systems try to recover the failing sub-systems previous correct status and process over the error.

If a failing sub-system is determined unrecoverable, the network must be able to shutdown that sub-system and and reconfigure itself if necessary. Status of the entire network must be maintained (which processors have which processes running on them, which processors have which IO devices attached to them). Then if one processor or process fails the other processes can ascertain the correct action needed to isolate the faulty sub-system and confine the error while allowing the remaining sub-systems to continue execution if at all possible.

External System Access

One of the first premises of this network was that it would provide a means for the minis to use the program development facilities of the CDC 6600 or other large scale systems. This connection to a large scale computer will be invaluable in providing source editing and cross compiling functions to aid the researcher using a mini. The network should provide a standard interface which can be used to communicate with a number of external computer systems. This interface should allow hookup to any desired external computer or system without causing major modification to the other network sub-systems.

The external connection interface must be able to isolate the communication protocol of the external system from the internal network protocol. It will act as a translator and queueing process between the two systems. In this way the other systems in the network can communicate with the external process in the same way they communicate with internal processes. This extends the virtual machine concept outside the confines of the mini network to include any outside computer. Any process running on the machine will still be separated from the knowledge of where other processes or devices are physically located in the network.

Network Performance Monitoring

The network itself is to be used as a research tool for the study of operating systems and networking techniques. Also, there will be many network sub-systems which must be tuned and refined to give the best overall operation of the mini network. Both of these require some way to measure how efficiently the network is working - performance monitoring. This must be a design consideration from the beginning and not an added attraction to be forced into the system later in its development.

The network must be designed with the appropriate "hooks" to allow all message traffic to be sampled to acquire statistics of such categories as:

1) Traffic to and from each processor

2) Traffic to each unique IO device and each class of IO devices

3) Memory usage on each processor

4) Transmission retries to any device or processor

5) Correlations between message traffic and presently running user jobs

6) User job statistics - duration of run, number of processors used by job, number and location of IO devices used, amount of time spent waiting for processor or devices

The above statistics are just an example to give a flavor of the areas which must be open to the performance monitoring function. If they are not made available by initial design, it could prove extremely difficult to devise a method of sampling these statistics as an afterthought.

The performance monitor function should be able to collect its data without appreciably affecting those statistics which are being gathered. This function must be able to run as an independent process without affecting those parts of the network performance which it is monitoring.

Standard Design Philosophy

This network is expected to survive many additions of both computer hardware and new software ideas. This requires that the network design allow a high degree of flexibility, a term which seems at first to oppose the term standardization, but it can be shown that standards can be maintained within a flexible structure. If all the previous requirements are to be molded into one integrated software system, there must be a standard design philosophy to adhere to. Without a single standard design for this flexible structure, too many idiosyncrasies of sub-systems will immerge and, combined with other sub-system differences, may start to drive the network efforts far from those requirements it was conceived to operate within.

The network will need a central system design to standardize the nucleus of the network virtual machine. This network nucleus must have standard communication protocols and data interfaces which allow change to occur within the standard format. An example of such a standard would be communication between network processes being strictly limited to messages. No global tables or flags would be allowed. While the processes must communicate through a message, possibly of some prescribed length and general format, the detailed inner format would depend on the process itself. As new processes are added to increase the functions in the network, there are many new internal formats for messages added. However, all communication is carried out by these same messages with no deviation from that standard.

The previous example illustrates the idea of a standard philosophy. Those major keystones which make the network conform to the requirements must not be compromised. There is a constant urge to refine and change things for a little more efficiency or ease of use. Requirements must constantly be reviewed at each point where a major addition is made to the network design. It must be assured that every addition conforms to the philosophy expressed by the functional requirements, and that the addition blends into the universal needs of the network instead of serving one particular subsystem to the detriment of the whole.

III. A NETWORK ARCHITECTURE

Introduction

The proposed network requirements have been outlined. It is known what the network goals are, but not how these goals will be achieved. This chapter will present an implementation architecture of hardware and software which will provide the required functions set forth in the previous chapter. The picture presented by this chapter is a broad brush treatment with no intention of presenting the minor details. It should give the reader an idea of the major architectural pieces that form the nucleus of the presented system.

There can obviously be as many different implementations as there are computer designers. The architecture chosen is a combination of the author's research of existing systems and the author's own experience with mini computer experimentation. This architecture will support the functional requirements and would seem to be well suited for the gradual implementation by successive follow-on graduate student efforts.

Hardware Configuration

While the hardware configuration of the network is a definite factor in the mini network, it will receive less emphasis here than the software configuration. The discussion of hardware will be limited to only the basic architecture needed to provide a suitable interconnection of the minis to allow the software system to operate. This section will describe the basic structure of the hardware used to form the network, relying on follow-on thesis efforts to perform the detailed specifications needed to construct and implement the actual hardware circuitry.

Network Bus Structure. One of the prime considerations of any computer network is how the various computers will be interconnected to form the network structure (topology). Much research effort has been spent on developing efficient topologies for numerous different types of networks. An excellent discussion of three of the most used topologies is given in Ref 9. It discusses the advantages and disadvantages of the three major topologies shown in Fig. 1. These topologies are used in most of the present networks where the computers are physically separated by large distances. While they work well for the mentioned configurations, these topologies were considered for the Digital Engineering Laboratory network and then rejected.

The mini network at AFIT will be located on one floor of the School of Engineering building and will probably not extend out of the building. Since the distances between the processors will be very small, a bus system can be used for communication. The bus structure (Fig. 2) allows all of the computers to be attached to one common set of communication lines. This structure allows any

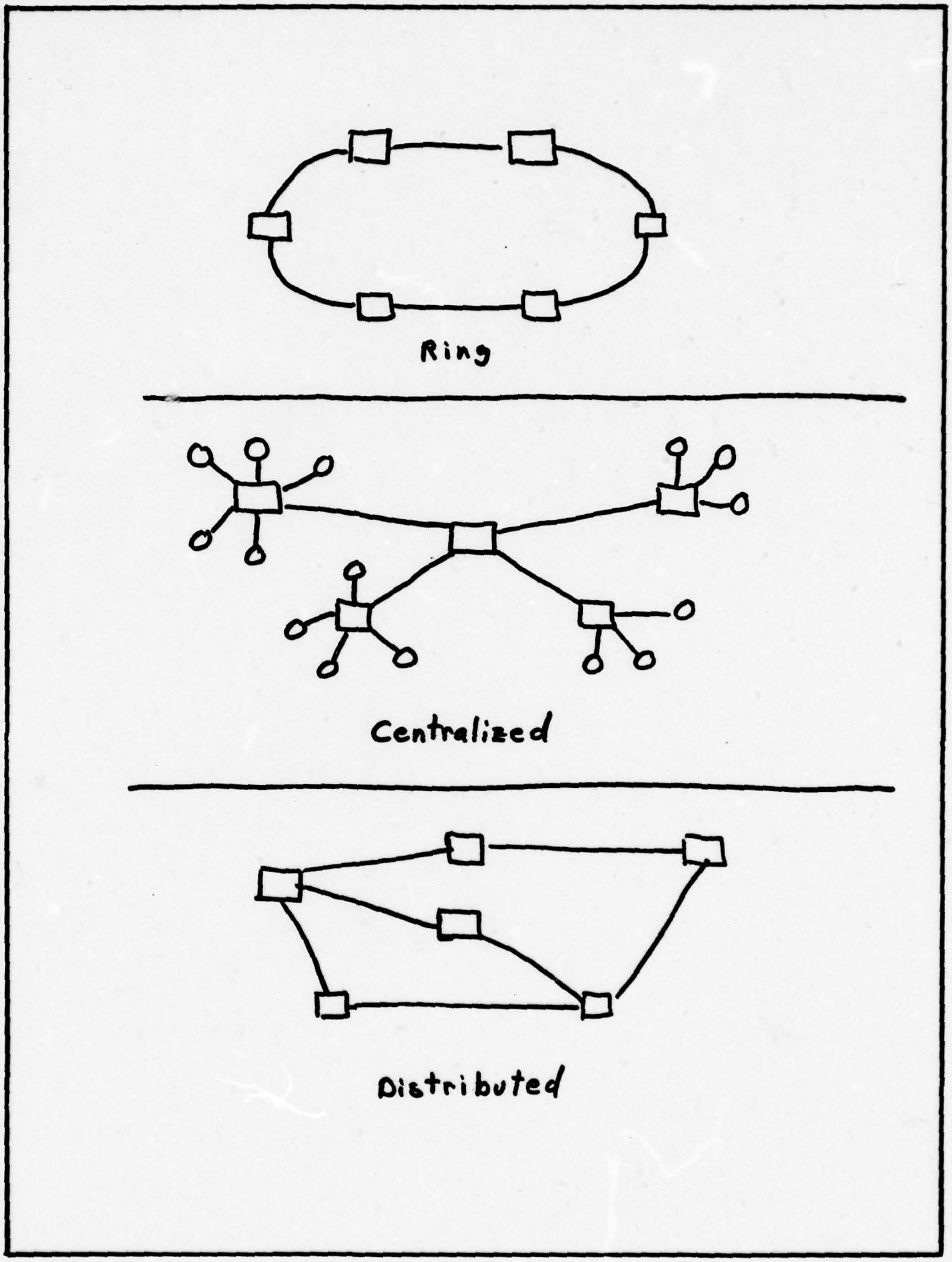


Fig. 1. Standard Network Topologies

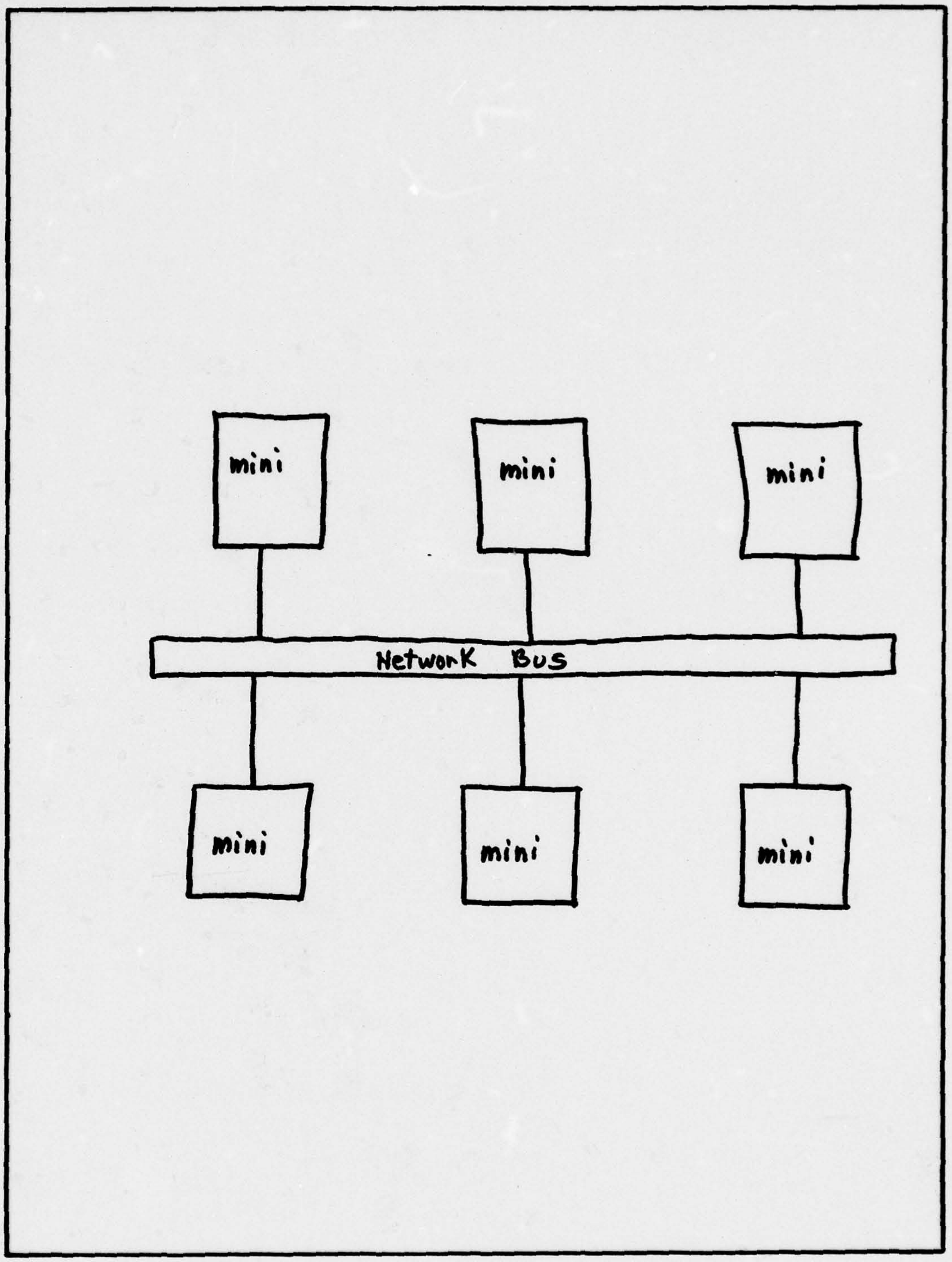


Fig. 2. Network Bus Topology

computer in the network to communicate directly with any other computer, but only one pair of computers can communicate at any one instance.

When large physical separation of the processors is not a factor, this bus interface structure has many advantages over the link structures of Fig. 1. The most important advantage is the flexibility of processor interconnection. There are no topology problems involving the connection of processors into the network. Every processor connects to every other processor in the network by simply connecting to the Network Communications Bus (NCB). When new processors are added to the network, one has none of the problems of the link networks such as: does this processor have satelite processors, or how many other processors must have a direct link to the new processor?

A second advantage is additional network reliability. When one of the network processors fails, the only effect to the network is the loss of those IO devices local to the failing processor. In some multi-linked topologies, one processor's failure might cause the failure of other processors which depend totally on the failing processor for their communication to the rest of the network. Since the bus structure allows all processors to communicate with all other processors, the failure of one processor can never cause another processor to be eliminated from the network's hardware structure.

The bus structure also simplifies the network's message handling software modules. Since there is only one connection from the bus to any processor, the processor need have only one input and one output queue to handle all message traffic. On the multi-link topologies the messages must be routed from one processor through one or more intermediate processors to reach the final receiving processor. With this structure the processors must have multiple message queues to store messages for forwarding to other processors down the line to the receiving process. Using the bus structure, messages are transmitted directly from the sender to the receiver with no intervening processors deciding where to queue and re-transmit the messages. The software queue handlers can be far simpler when using the bus for direct processor to processor communications.

There are disadvantages to the bus interconnection approach such as reduced performance. Since every message must travel on the same bus, only one message can be transmitted at any one instant. This would not seem to be a great hindrance, but it must be considered. The simplicity allowed to the queueing of messages by the bus approach would seem to outweigh any performance disadvantage.

The NCB can be implemented in either serial or parallel circuitry for there are advantages and disadvantages to both methods of operation. While this thesis does not intend to propose details of implementation, a suggestion is made on this point. The bus could be implemented with a double bus - one serial and one parallel. In this way performance studies and tradeoffs could be done and there would be a backup bus if one failed completely. This would allow for better performance and could allow for certain processors to have one bus as the primary with other processors having the second bus as the primary, thus separating traffic among the two buses (Fig. 3). It would greatly benefit research efforts to have both buses to provide comparison statistics as well as the increased performance and backup facilities.

Flexible IO Locations. One of the network requirements was to allow flexible IO device location. This is required to provide the flexibility needed to conduct all the various experiments which will sometimes want specific devices local to a specific processor. This flexibility also greatly facilitates addition of new IO devices and experimentation with load leveling of the processors by allowing easy relocation of any IO device from one processor to another. The third reason for wanting this flexibility is to redistribute IO devices in the event of a processor failure. If one processor with critical devices fails, the devices can quickly be transferred to another processor and the network restarted.

All of these reasons require that the network IO devices be fitted with a standard interface which can be attached to all the processors in the network. The different processors have their own logic circuits which differ from company to company. These circuits must be connected to an interface which is company dependent on

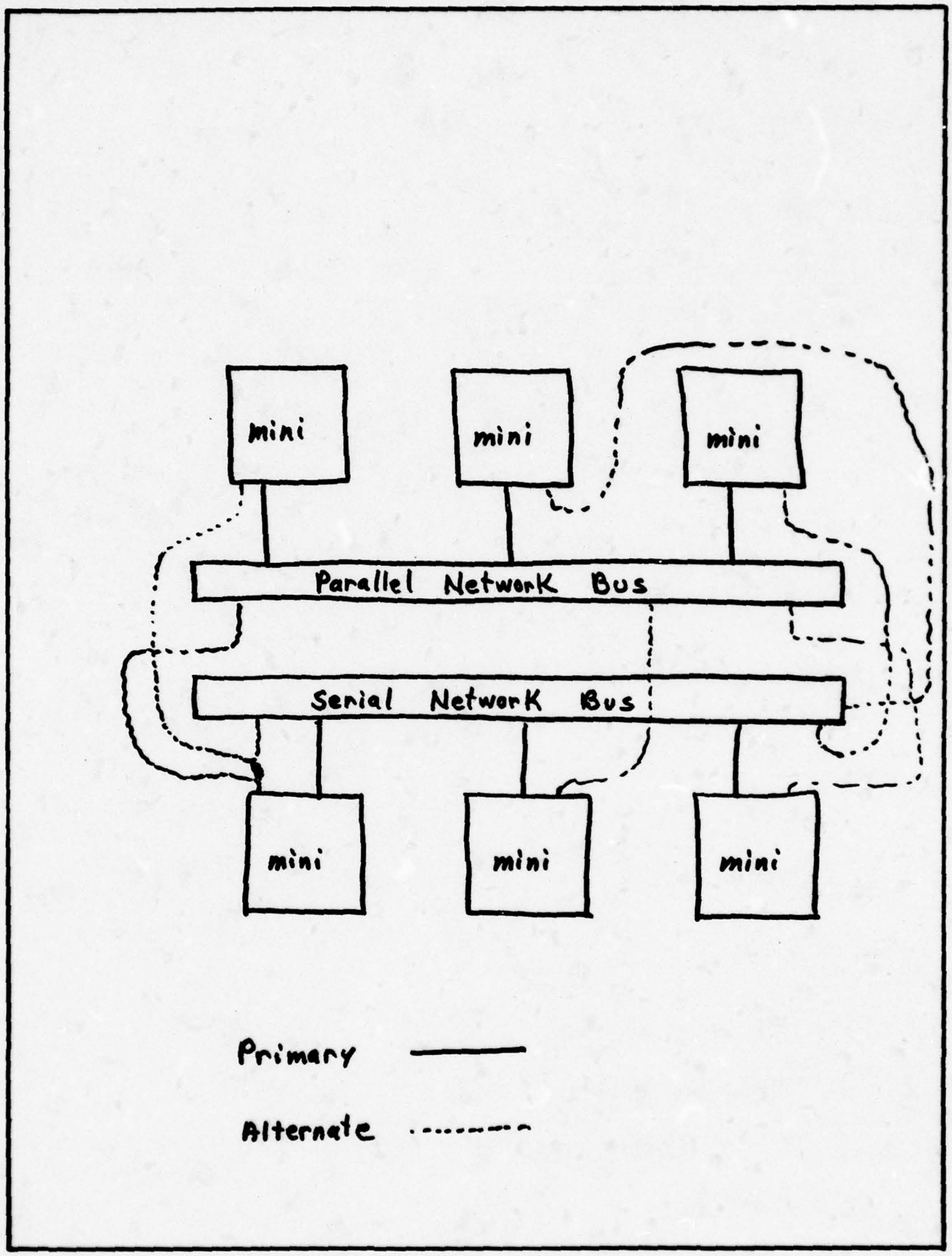


Fig. 3. A Double Bus Topology

one side but standard across the network on the other side.

Two standard interfaces as explained above must be designed. One should provide a standard serial protocol similar to RS 232 logic. This should not be a problem since all of the present minis in the network have such an interface. The interface, however, should provide for programmable switching between a wide range of baud rates to allow for the variety of devices which might be run through the serial interfaces.

The network processors must also provice a parallel interface of at least 16 bits wide (the wordsize of most of the minis memories) to allow another interface for high speed devices in the mass storage category (disk, tape drives). These interfaces might at first be limited to certain prime processors, since they would probably be more complicated and not easily available like the serial interface. The parallel interface would be the topic of one of the follow-on thesis efforts. After adequate test and perfection, each of the processors in the network would ideally be fitted with at least one parallel high speed mass storage interface so that these mass storage devices would not be confined to prime processors and could be transported like the slower serial devices.

Micro Interface. The minis will not be the only processors in the network. Of course, they compose the major computing power but the network will also have the capacity to use the microprocessor computing power for special functions and research efforts. These microprocessors are rapidly entering the research area and have many good applications in the network. These micros should interface onto the network bus in the same manner as the minis.

These micros will be used for special system functions and as research tools. They will have the same priority and symbolic addresses on the bus as the mini computers.

Real Time Clock. The network software will need a time reference in order to perform many of its functions. Therefore, a real time clock (RTC) should be introduced into the network. Each processor could have its own RTC, but this would mean extra cost both in physical equipment and the overhead of coordinating all clocks so that they remain synchronized. A better way would be to have access to the RTC through the network bus so that all processors would interface to the same RTC and all processors would be on the same time synchronization.

This access to a single RTC would be implemented by allowing the clock to generate interrupt pulses along the network bus and allowing each processor to keep its own count time. This count time would then be checked periodically to make sure all counts were synchronized. This would provide an integrity check to assure that all processor interrupt systems were operating properly.

Software Configuration

This section will present a general overview of the network software architecture design. This software architecture will fulfill the proposed requirements and should provide the necessary insight to allow follow-on thesis efforts to implement the detailed design and produce the actual network software. The proposed architecture differs greatly from most present operating system structures. Consequently, the design will not incorporate any of the operating system software provided by the manufacturers of the mini-computers in the network. The purpose of the proposed network is twofold. First, to produce a network which will aid the users of the Digital Engineering Laboratory computers, but, just as important, to allow students to participate in its design and implementation, providing actual experience in computer systems operating system design.

The Basic Network Structure. The proposed network can be classified as a distributed computer network. This implies that the network supervisory functions, as well as IO devices and data files, will be spread among the various network processors instead of being centralized into one prime processor which would control other processors as slaves. Every processor in the network will act as an independent entity. It will maintain control of its local resources while providing a part of the total computing power needed to support network services and user processes.

The network supervisory software will create a multiprogramming environment on each computer or Network Processing Unit (NPU). This allows each processor in the network to support multiple user and system processes which will run with apparent concurrency. The supervisory system maintains each processes' status and resources while also providing interprocess communication, logical IO operations, and a network data base facility. While all these functions are provided by the supervisory system, they are actually implemented as independent system processes.

Each NPU will have a KERNEL hardware supervisor which will handle the hardware schedule for that one NPU. This KERNEL will contain only those supervisory services necessary for the basic scheduling of the processes for hardware execution time. These basic scheduling services will include only the handling of hardware interrupts, context switching needed to give each process machine execution time, and basic queueing of messages which provide the communication between all processes. This KERNEL becomes the one common denominator of network software which will be the same for all NPU's.

All other services; IO handlers, communication between NPU's, error recovery, and file management; will be delegated to independent system processes running much in the same manner as the user processes under control of the KERNEL. These system processes combine to form the virtual machine which the network presents to its user. Because each network function is implemented by a separate system process, only the functions used by each specific NPU are allocated to that NPU.

Fig. 4 shows a possible network configuration. Notice that each NPU has a KERNEL supervisor, but that each NPU has only the system processes needed to support its local resources and the specific user processes which run on that NPU. It is important to notice the differences between each of the NPU's system processes. NPU1 has no user processes at all, instead it is responsible for handling many of the global services which aid the entire network such as the Data Base Manager, the Configuration Control and the Performance Monitor. NPU2 and NPU3 have the most common combination of user processes and IO system handling processes. Notice that the first three NPU's have a communication process which allows inter-NPU communication, while NPU4 has no communication process. This shows how any NPU can be configured to run in a stand alone mode by simply logically disconnecting it from the Network Communications Bus.

Configuration Management. Fig. 4 shows that the network can take on many different configurations of NPU's and IO devices. It follows that the network must have a way of controlling or managing the configuration so that it knows which NPU's are available and which IO devices are local to the various NPU's. This configuration will be under control of one Configuration Process for the entire network. This process will communicate with all NPU's

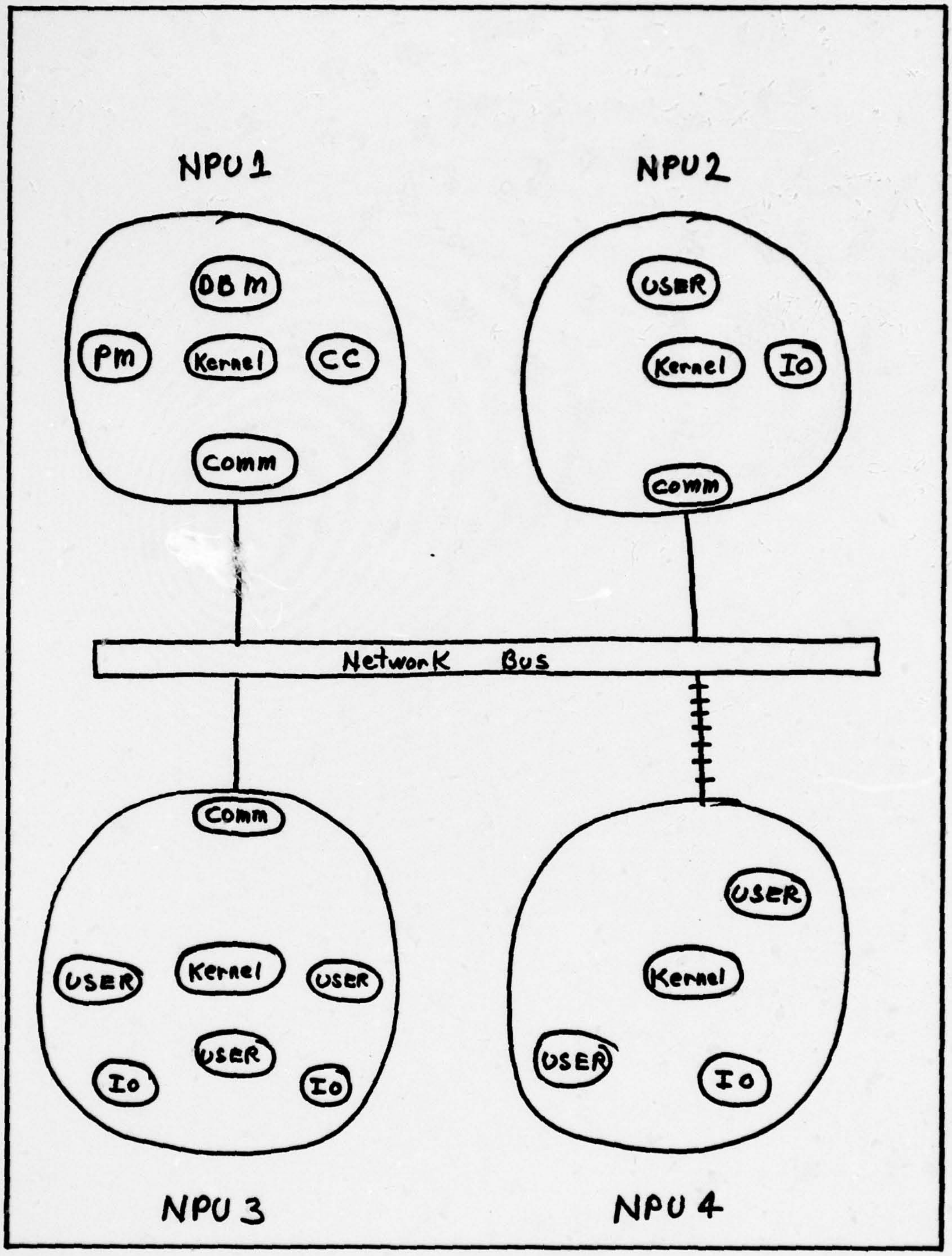


Fig. 4. A Typical Network Configuration

to keep track of which NPU's and IO devices are available and working properly.

When the network is started up, a configuration terminal will communicate with the configuration process. This terminal will allow an operator to configure the network as he wishes. The network has a prestored configuration table on mass storage which indicates the normal predesigned structure of the NPU's and IO devices. But, before the network system is started, the operator can change this table to indicate any change in the layout from the normal structure.

The configuration process then controls the loading of the proper software modules into the appropriate NPU's to accomodate the indicated configuration. This is not the end of the configuration control process execution. The process continues to run and communicates with all NPU's to check on the NPU status and the status of all IO devices. The configuration control monitors all network hardware and periodically checks to assure correct performance of the entire network. If an error occurs, this process can shut down any part of the network when the error is unrecoverable. The process takes on the responsibility of restructuring the network software at any time to allow for dynamic relocation of IO devices or network system functions. This allows the network to change its structure to accomodate errors or in response to an operator's commands.

Network Communication. This network must present the image to the user of being a virtual machine with many processors. To provide this image it must allow the user processes to communicate with each other without regard as to which actual processor a process is running on This communication is provided through a system of messages which are sent and received by the various processes. A communication's process at each NPU controls these messages and handles all inter-NPU protocol and error checking.

Since a process should not need to know other process locations, all messages between processes are sent by using a process name instead of a direct hardware address. Each NPU KERNEL then checks this process name. If the process is in its local process list, the message is queued for the process. If the process is not local, the KERNEL invokes the communication's (COMM) process which determines the receiving message's location and formats the message for inter-NPU transmission. The message is sent to the receiving NPU where another COMM process inputs, error checks, and queues the message for the receiving process local to the receiving NPU. All inter-NPU messages are handled by the COMM process, providing the standard interface between each NPU and the Network Communications Bus. The message handling is separated from both the rest of the supervisory systems and the user processes. In this way any change to the protocols used, or the methods of transmission used, will be isolated to the COMM processes and have no effect on the other network functions.

IV. PROCESS INTERCOMMUNICATION AND SYNCHRONIZATION

Introduction

The term "process" denotes the basic software unit of this network. This process implies a set of software ininstructions which are treated as an independent entity by the network, and as such become the network "atoms" or building blocks. All functions accomplished by the network are separated into these processes and it is the process to which the Network Processing Unit (NPU) processing time is scheduled. Processes do not denote user functions alone, but also denote many of the network service functions.

The network KERNEL's supervisory services will be limited to those services needed to coordinate these processes. The KERNEL, which will be detailed later, provides such basic services as hardware interrupt handling, scheduling of hardware processor time between processes, and interprocess synchronization. All advanced network services (file handling, user process scheduling, data base manager, etc.) will be implemented through actual system processes running in much the same manner as the user process. These system processes form the layers of the operating system above the network KERNEL which the user sees as the network virtual machine. Even some of the most basic functions of the operating system (memory allocation and deallocation) can be separated from the KERNEL and be implemented as system processes.

Process Types

The system processes mentioned above lead to the discussion of process typing. It was stated that the system supervisory processes which will form the network virtual machine will run much in the same manner as the user processes. This is true, but the key word is much, not exactly, in the same manner. While the system processes will compete for the NPU along with the user process, they will compete at a higher priority and have some privileges that the user processes don't have.

Looking closer, it can be seen that the system process must have a priority edge over the user process since the user process must depend on the system process for many of its services. An example is the handling of IO requests. Each IO device will be under control of a system IO process. If two user processes were running in one NPU, the following events might occur. User process one (UP1) might come to a point where it makes an IO request to the IO process. If both the IO process and the user process two (UP2) are in the ready state (both are waiting for NPU time), which process should be scheduled on the NPU first? Obviously, the IO process must be scheduled first in order to maximize IO and NPU concurrency. If the IO process is scheduled it will start an IO operation and immediately give control to the KERNEL, allowing UP2 to be scheduled while the IO is running. If UP2 is scheduled, the IO process must wait until UP2 is interrupted for some reason and then if there were more processes the IO process might not receive the NPU even then. This necessitates the separation of processes into System Processes (SP) and User Processes (UP) for priority scheduling on the NPU's.

There is also another consideration for the SP and UP division or classification. The SP must be allowed to run in a freer state than the UP. Such operations as START IO and TEST IO which affect the physical control of the hardware state must be reserved for the network virtual machine so that it can maintain control. It would not do for a UP to rewind a tape drive or disk that was being shared by other UP's under the control of another SP for all users would suffer.

Although many supervisor services are delegated to system processes, these processes are given the protection of being part of the virtual network machine by being classified as privileged or system processes and can be distinguished from the more limited user process.

Process Communication

The network described consists of a KERNEL hardware supervisor and a number of UP's and SP's all running on the NPU interacting with each other. They must communicate for the network to operate, but how is the communication carried out? Here, another basic requirement must be remembered. It has already been stated that many of the network operating system functions will be implemented as processors.

The network requirements also state that these supervisory functions will be separable so that they may run with apparent concurrency. This implies that both the system processes and all user processes cannot be allowed to use common memory storage as a communication medium, for they could not then be independent of each other nor could they be independent of each other's location in the network.

This prohibition from common memory as a form of process communication seems overly restrictive at first, but it must be remembered that the processes may not have any memory in common. They could be running on totally separate NPU's and one of the functional requirements states that this communication must not depend on the location of either process. The processes must be able to communicate with processes on and remote to their local NPU's, an attribute which makes the use of global tables impossible. Instead, all communication between network processes will be through a system of messages. These messages will be under the processes control and will make up all the communication between any two independent processes.

One can see that these messages allow a total freedom of process location. It now makes no difference if the receiving process co-resides with the sending process; for if not the network will automatically format the message for transmittal to the remote NPU. At the receiving NPU, the message is re-formatted by the network to its local form and presented to the receiving process just as it would be if sent from a local process.

The process message becomes the standard information transfer interface between all network processes whether they are classified as user or system type.

All processes are restricted to message transmission for all data and control communication. This actually provides more freedom rather than being truly restrictive. If a user program can be designed into processes which can be run concurrently, this non-common communication restriction actually benefits the program if enough NPU's are available to split the processes among the NPU's. Here the user has an inherent design tool which will allow concurrence if it is available, but will not hinder the program if all processes must run on one NPU. If the program had been designed using common memory communication, the availability of the second NPU could not be used to promote the program's inherent concurrency.

The message communication requirement is carried throughout the network's system processes. Whenever a user process calls for the execution of an IO operation, it is really sending a message to the IO process handling that IO device. This message is transmitted to a remote NPU and IO process if necessary without the user processes' apparent knowledge. This goes for all system processes. No system process shares a common memory data structure with any other process. This allows maximum flexibility for location of system processes used by more than one NPU such as a data base manager. All processes, user and system, may be sending requests for data base action from all parts of the network. All requests are then transmitted to the data base process wherever it is located. The message interface also allows for ease of replacement of system processes, since there is no common memory linkage. The interface between separate functions in separate processes is clearly delineated through the message formats between associated functions or processes.

Process Synchronization

The processes now have a standard method of communication; they can send messages to other processes and presumably receive messages from other processes. This solves the problem of getting data from process to process, but does it totally solve the whole communication problem? For example, process UP1 needs to write a line to the line printer handler LP1. If UP1 simply sends one message to LP1 there may be no problem if LP1 is also expecting just one specific message from UP1 (Fig. 5). In reality, however, UP1 will probably run in a loop sending many lines to LP1, and LP1 may be receiving other messages from other processes (Fig. 6).

It can easily be seen that some form of control or process synchronization must exist for any communication effort to be successful. Obviously, LP1 must be able to wait until it receives a valid message or its execution

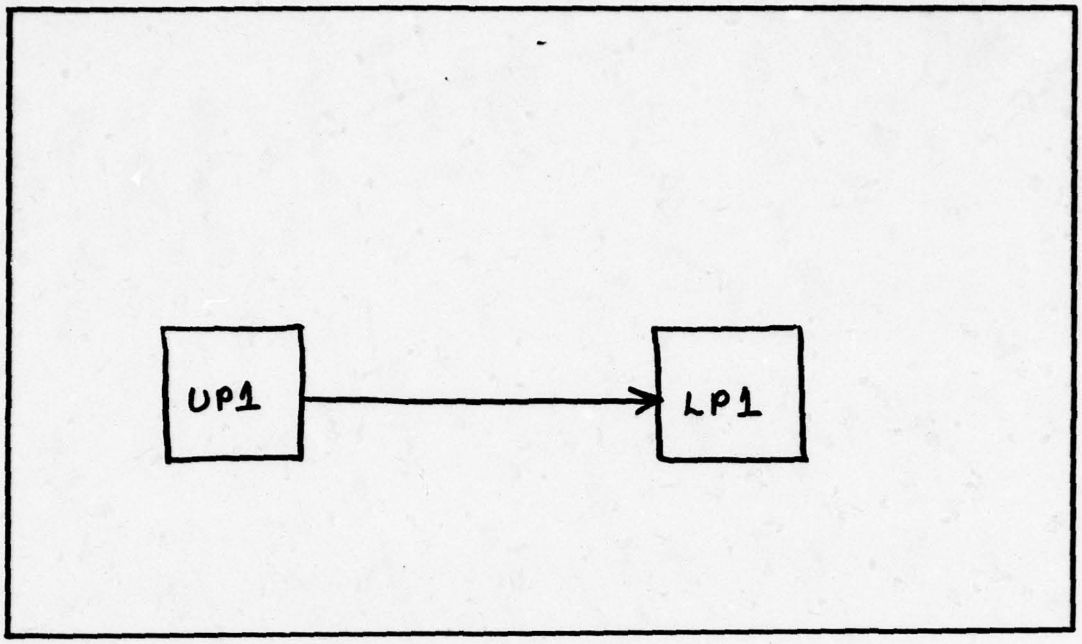


Fig. 5. Simple Communication

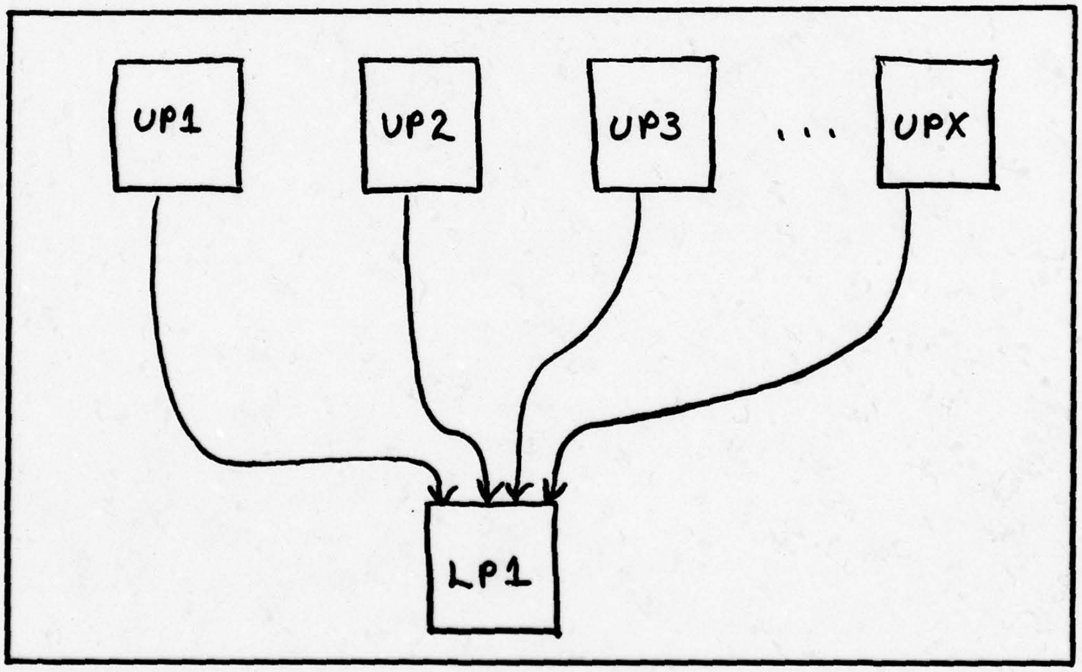
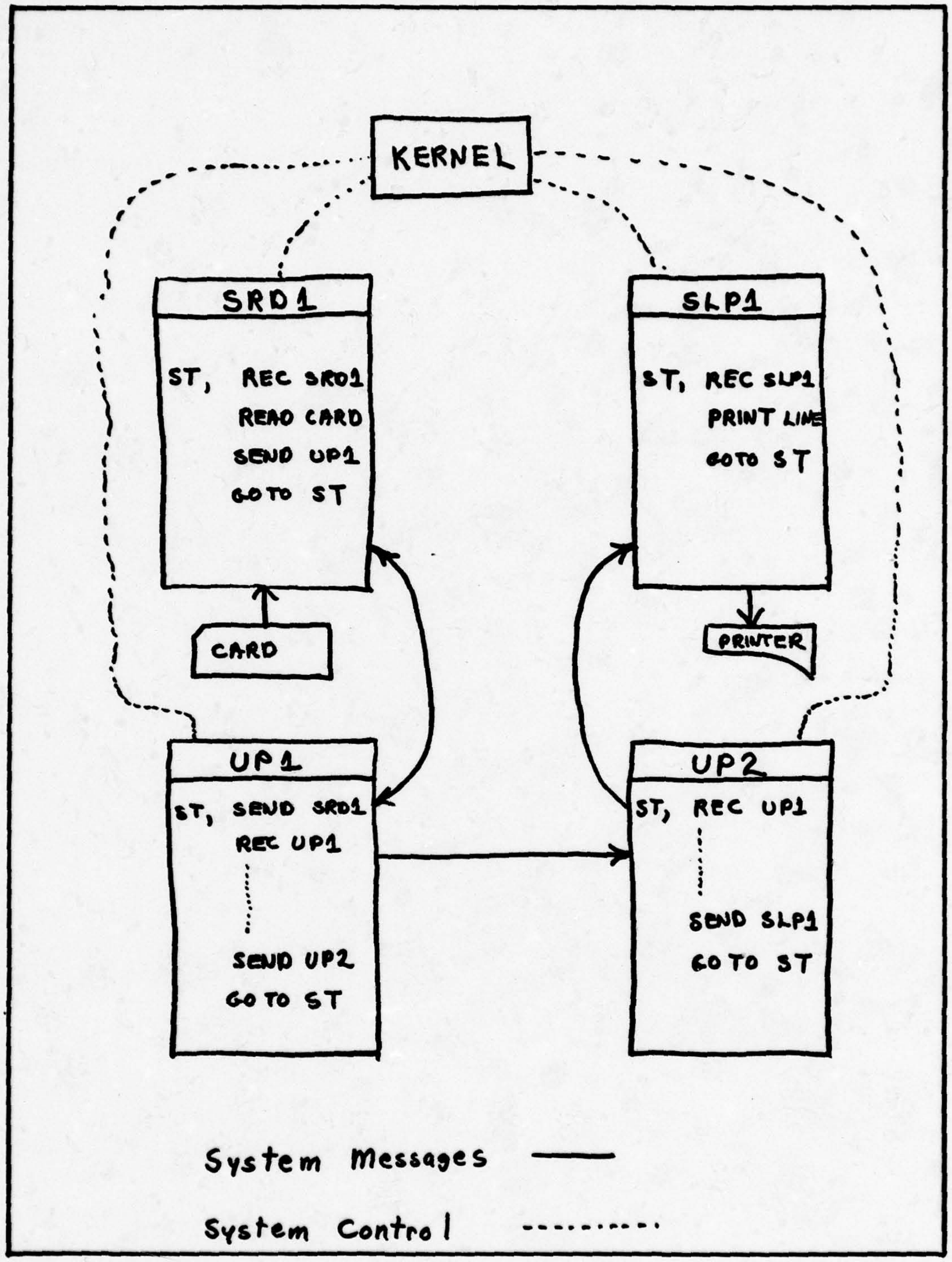


Fig. 6. A Synchronication Problem

loop would be spouting endless garbage from memory to the line printer. Not so obvious at first, but just as important is the fact that UP1 might want to wait in its loop for LP1 to signal that the print operation was successful before UP1 continues. If UP1 were reading a card instead of printing a line, it would be forced to wait for the reader process to return a message with the card's contents.

This process synchronization is one of the prime responsibilities of the KERNEL supervisor. It must be capable of controlling the processes and allowing them to specify when they need to wait for an incoming message or when they wish to send a message. E. W. Dijkstra, (Ref 6:253) has provided a solution to this synchronization problem through the use of his P and V counting semaphores when used together with basic message queueing for each process. A good explanation with example is given in (Ref 6:254) so a total explanation is not attempted within this paper. Instead, an example is given (Fig. 7) showing how the processes would use the Send and Receive primitives (Ref 6:254) which combine the P and V semaphores with the queueing of the process messages.

This example presents a simple user program which must read a set of cards, do some form of conversion, then present the converted cards to another module which will format the converted cards into a report and output the report. This example also shows the advantage of splitting a program into independent processes. One process can read the input and



```
SRD1
ST,  REC SRD1
     READ CARD
     SEND UP1
     GOTO ST
```

```
SLP1
ST,  REC SLP1
     PRINT LINE
     GOTO ST
```

```
UP1
ST,  SEND SRD1
     REC UP1
     :
     SEND UP2
     GO TO ST
```

```
UP2
ST,  REC UP1
     :
     SEND SLP1
     GO TO ST
```

Fig. 7. A synchronization Example

convert it, while a second process can concurrently process the converted input and print the desired report. The intermediate converted cards will be passed between the two processes through the use of the process message.

Shown in Fig. 7 are the two user processes running with the two system processes SRD1 (card reader IO) and SLP1 (line printer IO). When the network was started the SRD1 and SLP1 processes were automatically initiated as necessary processes to this NPU through the configuration control. Both SRD1 and SLP1 accomplished necessary startup code then executed REC primitives to notify the KERNEL that they will wait until some process sends one of them a message. The two user processes then enter the NPU and begin execution. Either process could receive control of the NPU for execution first and either way the processes will execute properly due to their SEND and REC primitives.

Assume UP1 executes first; it will immediately send a message to SRD1 asking it to read one card. Both the SEND and REC act as supervisor calls (SVC) to the KERNEL, at which time it may assign the NPU to the highest priority process ready for execution. SRD1 is now scheduled which in turn initiates a read which allows the KERNEL to schedule another process. If UP1 is scheduled it then executes a REC UP1 which makes it wait for a message. If UP2 is executed it executes a REC UP2 which causes it to wait.

Now all four processes are waiting - three (UP1, UP2, and SLP1) on REC message primitives, and SRD1 is waiting for the hardware card reader to complete its operation. When the read operation completes the KERNEL schedules SRD1 which executes the SEND UP1 to send the card contents to UP1, SRD1 then executes the REC SRD1 to again wait for the next message or command. Up1 now has a message to work on and the KERNEL schedules it for execution. UP1 does its processing of the card data and formats it into a message to UP2, which it sends with SEND UP2. When UP1 gets the NPU again it will immediately execute the SEND SRD1 to read another card then wait for it with the REC UP1.

Now SRD1 is waiting for a card being read, UP1 is also waiting for the card, but UP2 has a message to process - the one UP1 just sent. When UP2 is scheduled it takes the message and uses the data to begin formatting the first line of a report. UP2 then executes the SEND SLP1 to send the line to the line printer process. After sending the message to SLP1, UP2 loops back to REC UP2 to receive the next message. SLP1 will then receive its message, output a line to the printer, and loop back to wait for another line to be printed.

So the processes continue, all four processes running apparently concurrently, but yet in perfect synchronization. Each handles the part of the program for which it is designed, then passes its data along to the next process in line.

Through the simple primitives of SEND and REC, which combine the P and V operations with message queueing, the KERNEL is able to schedule each process when it has work to do (a message to process). Therefore, the processes can not only communicate with, but also can control, each other's timing all through the use of the interprocess message.

V. BASIC NETWORK PROCESSES

Introduction

It has been shown that most of the network's virtual machine will be implemented in a special class of processes identified as Network System Processes (NSP). These processes are the separable supervisor functions specified in the functional requirements. While the proposed network will eventually develop new NSP's as it grows and changes to meet new requirements, this chapter outlines those basic NSP's which are needed to the initial network configuration. A brief explanation of what each required process is expected to handle will be given. It is to be remembered that while the NSP requirements are outlined in some detail, this is not an attempt to give actual detail design specifications for each process. Follow-on implementation should further research new ways to accomplish the process function while staying within the proposed basic architecture and requirements.

The Supervisory KERNEL

The KERNEL is actually the most basic software unit in the network. Technically, it is not a system process, but is the scheduler of all system processes and, as such, the implementer of the network must know what is expected of it. This piece of software handles only those functions integral to context switching between the network processes and the handling of the hardware interrupts. It should not be allowed to expand into functions which could be pushed up into the NSP layer. A good example is memory allocation. One would first think that the KERNEL must handle the allocation of the NPU central main memory; if not, how would a process ever be allocated memory to run in? This problem can be solved by preallocation of memory to an allocation process at network initiation, so that when the network KERNEL is loaded the allocator process is also loaded. This allocation process will be described in more detail later.

The prime function of the KERNEL is the scheduling of the NPU to those processes initiated on that particular NPU. Each NPU will have exactly one KERNEL associated with it. Since the KERNEL is not a process, and must communicate only with those processes physically located in the same NPU, it is allowed to communicate through central memory in the form of SVC (Supervisor Calls) instructions executed by the NPU hardware.

While the minis in the network may not have a SVC instruction as such, one can be implemented on almost all minis through the use of an illegal execution of some form of instruction. It is desirable to implement such an illegal instruction trap, so that whenever the pseudo SVC is executed the hardware itself performs the entry into the KERNEL. This provides better control of the hardware than allowing the processes themselves to branch into the KERNEL directly and possibly allowing them to branch incorrectly, causing malfunctions in the KERNEL due to a using process. The KERNEL's scheduling integrity should be protected at all costs for should the KERNEL fail, the entire NPU is likely to fail.

The process now has the means to communicate with the KERNEL, but what requests can it ask from the KERNEL? The follow list of KERNEL SVC's is suggested for the basic network implementation:

ACS - add a process to the KERNEL's scheduling list

RMS - remove a process from the scheduling list

XMT - send a message to a process

XWT - send a message and wait for reply

REC - receive a message from a process

While other SVC's may be added for extended versions of the network, these will allow a basic configuration KERNEL to control the processes on a NPU. The ACS and RMS SVC's add and delete processes to and from the KERNEL's active process queue. They can only be executed by a system process and have no effect when executed by a user process. The XMT SVC allows one process to send a message to another process and remain ready for execution, while the XWT SVC informs the KERNEL to send the message and place the process in the wait queue until receipt of the message by the receiving process has been verified. The REC SVC tells the KERNEL that the process wants to wait for a message. If there is no message queued for the process, the KERNEL puts the process into the wait queue, otherwise the message is made available to the process and the process remains in the ready queue. This small set of SVC's handle the basic communication needs between the KERNEL and its processes. The implementation format of the SVC is left to the implementer, but a possible solution would be to have all parameters for a particular SVC follow it in sequential words in memory, or have a list of parameter addresses follow the SVC in memory.

The KERNEL must be able to receive the hardware interrupts and notify the appropriate system process which is handling the device that caused the interrupt. The hardware interrupts can be handled by converting them into messages and putting them into the appropriate queue for the proper handling device. In this way there are no special communication methods established between the KERNEL and the system IO handlers. The system handlers simply establish a message port through which the KERNEL can signal the presence of an interrupt and the interrupt is processed much in the same manner as a message. It should be noted that since the KERNEL controls the priority of execution of processes, there should be no timing problem in this method of handling interrupts. Whenever an interrupt occurs, the KERNEL will know which process it belongs to and schedule it as the next process to run after entering the interrupt as a message in that process' queue.

Memory Allocation

It was stated earlier that resource allocation such as memory allocation can be done by one of the system processes. The memory allocation system process is responsible for coordinating all requests for memory throughout the network. This memory allocation can be handled in one of two ways. A process can be assigned to each NPU to handle only the memory allocation for that NPU, or one process can be assigned to handle memory allocation for the network. Both methods should be experimented with to see which best serves the needs of the network.

The local allocation process will be loaded with the KERNEL at nework load time and will be responsible for allocating the memory for only those processes running on the same NPU. At start up of the network the memory allocator, along with other system processes, will be preallocated for that NPU. From that initial point on, the allocation process will control all requests for processor central memory. Whenever a process needs memory for buffers, tables, or to load another process, a message is sent to the allocation process. The allocation process handles the messages, updates its memory allocation tables, and either rejects the allocation request or honors it, then sends a reply to the process indicating the result.

When a process needs a segment of memory, it would send a message of the format.

MEM	CON	AMOUNT	ADDR

MEM - is the symbolic name of the memory allocation process

CON - is the request command to the process - either

ALLOCATE - request a new segment of memory

DEALLOCATE - return an old segment of memory

AMOUNT - indicates the number of words being requested or returned

ADDR - used only with a return request to indicate which segment of memory is being returned.

The allocation process then returns a similar message of the same format indicating in CON whether the request was granted or not. By creating a process to handle memory allocation, the KERNEL is relieved of this function normally integrated into the basic machine supervisory software. The next logical step for network-wide memory allocation could be taken.

A good experiment might be allowing one Memory Allocation Process to handle the entire network's memory allocation. This process, running on one NPU, would be responsible for the memory on all NPU's. This virtual allocation would have no effect on the other processes, since they request memory through the interprocess message system. Obviously, the allocation of memory on NPU's remote to the NPU where this allocation process is running will take a little longer, since the messages which control the allocation will have to travel between NPU's. This is really of no major consequence, however, for the allocation and deallocation requests do not occur frequently within a process. Most allocations will be executed during process initiation and termination with relatively few requests executed during the process life.

It is not necessary to limit the allocation completely to either local or global means. The network could have a mixture of both global and local memory allocation. An example could have one NPU with a high occurrence of memory allocation requests best handled by the local allocation, while the remainder of the network NPU's had a lower occurrence of memory requests which would be handled adequately by one global allocator.

Two configurations of the network memory allocation processes are proposed to handle the network needs. First, the memory allocation process could be configured to run on the NPU with the large number of memory requests. In this configuration that NPU's requests would be local requests, which would be handled speedily, while other NPU requests would come over the network communication system and would be handled less rapidly due to the inter NPU traffic.

Second, two allocation processes could be running on the network, one on the busy NPU to handle only its memory requests, while a second allocation process would handle all other memory requests for the remaining NPU's in the network. This example demonstrates the benefit provided by separating this system function by the use of inter-process messages. The entire spectrum of using one process for each NPU to using one process for the entire network and all steps in between become feasible.

The network should initially be designed with only one allocation process to free memory in the remaining NPU's for other processes. If later development shows that certain NPU's, or all NPU's, should handle their own memory allocation, then the processes are added when needed. In any case, no other processes are affected, either way they still generate their requests for memory in the same protocol without regard to where those requests are actually processed.

User Job Scheduling

The KERNEL and its supporting system processes are in the NPU and waiting for work to do, but there must obviously be a user process in the system before the system is actually used. This section will explain the basic User Job Scheduling Process which will handle that interface to allow a user to introduce his work into the network.

There will actually be two different ways to enter a user job into the network. The job could be entered as a batch job through one of the batch IO devices or interactively through one of the terminals. The network will be provided with at least one batch terminal which will allow the user to submit jobs through the use of card decks or paper tapes. This batch system should be able to use job control commands to allocate resources to individual user processes in a job. This job scheduler must interface with the other system processes to provide the necessary batch facilities. Most of the jobs will probably be entered through the use of interactive terminals. The jobs entered through the terminals will be handled much in the same way as the batch jobs except in an interactive procedure. The batch and interactive procedures should accomplish the same functions while allowing each to do it in its prescribed mode.

This user scheduling function should actually be broken into three separate functions with one process per function. A User Command Process will be used to standardize the allocation process for both interactive jobs and batch jobs. This process will receive commands from the user job through the Batch and Interactive Interface Processes. The command process handles all user commands uniformly regardless of from where they are issued.

This process will handle requests for memory for initial scheduling of a user process, requests for IO resources, requests for a particular NPU or class of NPU's to run on, etc. Any command which the user can present to the network will be coordinated through this User Command

Process.

Although the User Command Process actually handles the execution of user commands, different formats and responses are required for the batch needs compared to the interactive needs. The Batch Interface Process upon initiation will request allocation of those devices designated as batch devices by the network configuration tables. It will then present a read request to the batch input devices and wait for any input to occur. Upon the presence of batch input, this process converts the input command for presentation to the command process.

The Batch Interface Process also handles output to the batch devices from the batch user processes. Along with input and output control, this process will establish communication to the network operator console through the configuration process. This process is the central control point between all user processes running in the batch mode and the rest of the network system processes.

While the batch system handles all the batch devices, the Interactive Interface Process handles all the interactive terminals on the network. Since there is much dialogue in the interactive mode, this process acts as the go between to the User Command Process. It handles the logon procedure and prompting messages for all the interactive terminals. Its primary job is to interface with the User Command Process and convert the interactive user commands to the internal format to be processed by the command process.

There are many advantages to the logical division of user job scheduling into these three parts. The main advantage is the consolidation of both the interactive and batch command processors into one processor. By using the interactive and batch interface processes, all user processes can be scheduled by the one command process, eliminating the duplication of functions that might be done to schedule both types with separate full scheduling processes.

Local IO Process

The Local IO Process handles all the functions that an IO driver routine would handle for a large mainframe computer supervisor program. This includes the physical handling of the device, the status keeping for a device, and the queueing of requests for a particular device. There will be one Local IO process for each IO device, or class of IO devices, in the network. Whenever a process executes a read or write to an IO device, it is actually sending a message to a Local IO process somewhere in the network.

When an IO message is issued, the KERNEL determines whether the message is to be a local device or one located on a remote NPU. If the request is not local, the Communication Process (COMM) is invoked to send the IO request to the appropriate NPU. There a second COMM process queues the request then sends it to the Local IO process in that NPU. Once placed in the local process queue, all IO messages are handled in the same way by the Local IO processes on the various NPU's.

The user process IO messages are initially queued by the KERNEL in a first-in-first-out method. The IO process can then input the messages and queue them in any manner. If one type of IO device is best handled with a priority queueing scheme, then the process handling that device will maintain a priority queue internal to itself after receiving the messages. Note that all aspects of the IO device manipulation, such as the type of queueing, remain inside this process to isolate the device from the network. If an experimental type of queueing is to be tried, the IO process is replaced by the experimental process and the network sees no change external to the changed process.

The IO process will have a second message input port separate from the IO request. This second port receives interrupt messages from the KERNEL. While the KERNEL handles the actual interrupt, the interrupt is immediately passed to the handling IO process for actual logical processing. The KERNEL is merely a switch to route the hardware interrupt to the IO process which then services the device according to its device dependent needs. All physical manipulation of the device is initiated within the IO process. It must consequently run as a privileged system process. The entire process will not run in the privileged state, but as a privileged system process it can request to run in the privileged mode for short execution sequences such as the issuing of a START IO or TEST IO operation.

The Local IO Process is also responsible for maintaining the status of its IO device. It will periodically check the device to make sure it is operating properly. If the process detects a constant error condition on the device, it will then send an error report message to the Configuration Process discussed later. The Configuration Process will also periodically send check messages to the IO process to verify that the process itself is running correctly. These status messages help the entire network to monitor itself and detect errors at the earliest possible time to prevent loss of additional functions due to the failure of other functions.

VI. EXTENDED NETWORK PROCESSES

Introduction

The previous chapter described the basic processes needed to form the network software required to operate one NPU only. There was no provision for any configuration control of the NPU's and IO devices nor any system process to handle inter-NPU communication. These system processes were not included in the basic processes for it is the author's opinion that the basic functions must be tried on a single NPU before they can be extended to encompass the implementation problems which will accompany the integration of multiple NPU's into the network.

For this reason some of the processes which are basic to the network as a whole, but not an integral part of a single NPU operation, have been left for this chapter. This chapter will describe the remaining system processes which the author feels must be included in the network to meet the functional requirements given in Chapter II. This does not mean that other processes will not be implemented later to provide additional function for the network, but the lack of any of the following system processes would cause the network to fall short of the requirements presented.

Configuration Management

Even though the proposed network will distribute supervisory functions among all the NPU's, there must be a control point to orchestrate this distribution. The Configuration Management Process will be this central control point. It will maintain the network status for all NPU's and IO devices within the network. This status will form a structure describing which NPU's the various IO devices are attached to and which supervisory service processes are running in each NPU. From this status it can monitor the entire network to make sure that all parts are running properly and take proper actions when failure occurs.

The configuration process will also communicate with a master network console, an interactive terminal of some type. Through this console the network operator can determine what is happening in all parts of the network. The operator must be able to start and halt the network, as well as reconfigure it by adding or substracting NPU's and IO devices. The network will have a predesigned structure stored away before the network is started by this operator.

When the network is first initiated, the operator will load the configuration process into one of the network's NPU's. The process will input the predefined structure and then communicate with the master console. At this point the operator will have the option of starting the network in the preset configuration or of making changes. He can use the console to reconfigure the network to allow for minor differences for this running period. On a certain day one NPU may be left out of the network for a special stand alone experiment, or various IO devices may be out of order and not available to the network.

After the configuration is set the operator would start the network. The configuration process then starts to load the specified software into the NPU's through a bootstrap loading system. The software is loaded according to the Network Status Table; only those system processes needed by each specific NPU are loaded. All necessary system processes are loaded, given independent life, and the network is up and running.

Once the network is running, the configuration process takes on an additional function. Throughout the life of the active network, the process constantly sends consistency check messages to all NPU's and all IO devices in the network. It periodically sends a message to each device handler to assure their proper operation. If a correct response is not received within a predetermined time interval, the configuration process assumes a problem exists and begins to attempt isolation of the problem area.

The isolation of the error in this case is no simple matter. Fig. 8 shows that the error between the configuration process and the IO device being tested could have occurred in any of six software modules (including the configuration process itself) or four pieces of hardware. The configuration process will systematically check each piece until the problem is found. If a problem is found in the software, the configuration process will try to

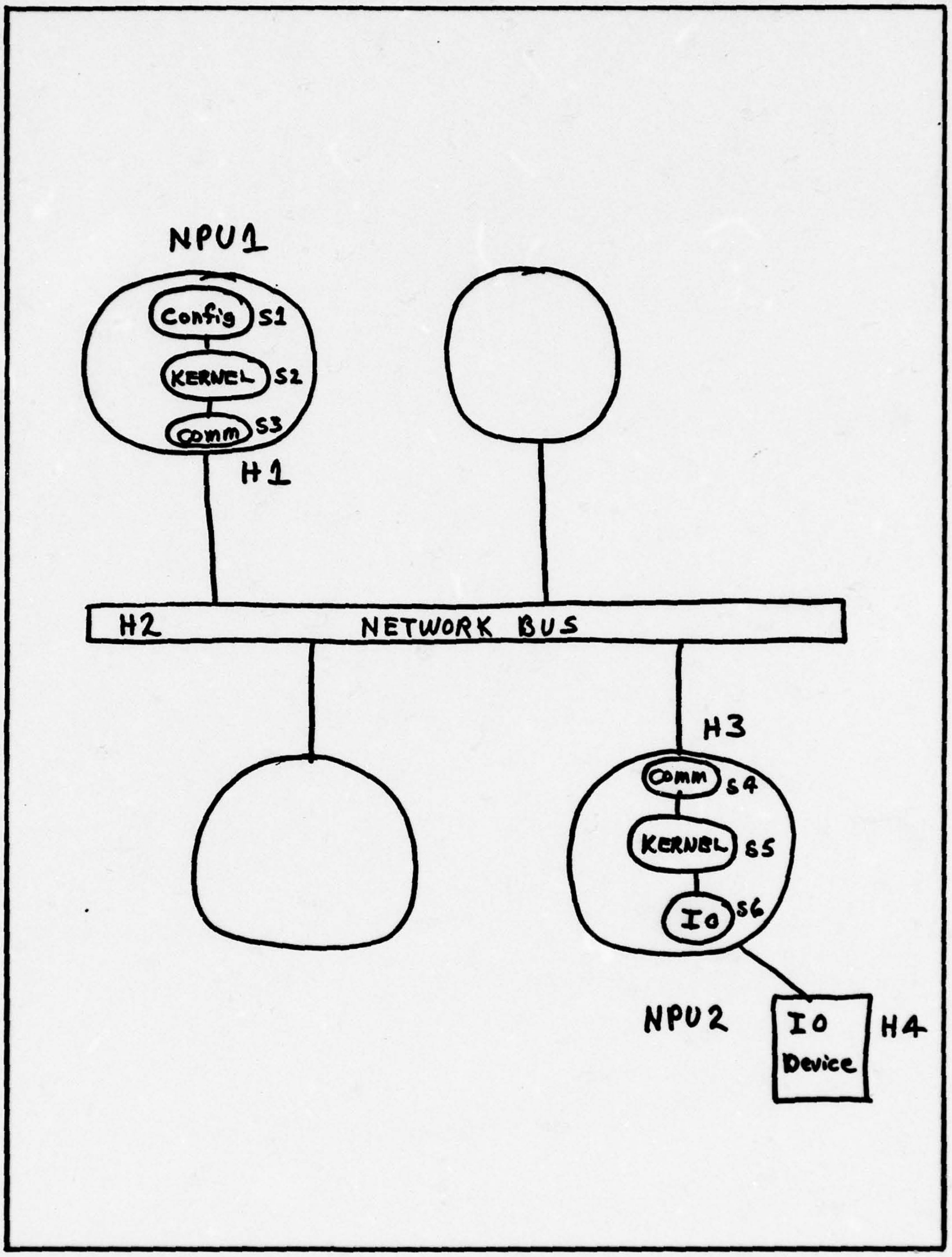


Fig. 8. An error confinement problem

correct by reloading a backup software module. If the problem cannot be solved, the failing part of the network will be marked inoperable and will be logically disconnected from the network.

The Network Status Table also keeps track of where all the processes are running in the network and where global network data files are located. When a process asks for use of one of the global network resources, the configuration handles the initial request and controls access to the resource. This allows the configuration process to keep the status of all resources in the network, not just the hardware status. This also implements the global property of resources by allowing a user process to find the resource without knowledge of its network location. It simply requests that the configuration process establish the linkage for further communication with the requested resource.

It is obvious that the configuration process is the hub of the proposed network. Without its central control, many of the virtual features of the network, like process independency and common environment, would be lost. Due to its importance, the configuration management process and its host NPU must be reliable and always available to the network. To help accomplish this goal another specific implementation point is suggested.

It might prove advantageous to dedicate one microprocessor, along with one interactive terminal, to this function. Together they would act as an intelligent terminal connected directly to the Network Communications Bus, which stands alone - separate from all other network NPU's. In this way if any other NPU's fail, the configuration control is not lost, and if the microprocessor failed, it could be replaced by a backup micro. This also prevents any other processes from running along in the same NPU with the configuration control and possibly causing a failure to this critical function. This is simply an implementation suggestion and, by no means, a required configuration.

Interprocessor Message Management

This section explains the processes which actually accomplish the communication between the NPU's that make up the network. Whenever a process makes a request to send a message to another process, the KERNEL first receives that request. The KERNEL then checks its active process queue to determine if the message is to be sent to a local process. If the process is found in the active queue, the KERNEL queues the message to that process and updates the process status if necessary. For the local process that communication is complete and under the control of the receiving process.

However, what happens when the KERNEL does not find the process local to its NPU? The KERNEL knows the message must be to a global process which is out of its control. It queues the message to the Communication System Process (COMM), and schedules COMM for the next execution. COMM then takes over and performs the necessary steps to transmit the message to the appropriate NPU. How does COMM know which NPU to send the message to? Let's look at how COMM is started and what information is available to it.

When the network is first started, every NPU is bootstrap loaded with at least the KERNEL and the COMM system process. The COMM process must be loaded along with the KERNEL, for until it is present each NPU is unable to talk to any other NPU. When activated the COMM process tries to establish communications with the Network Communications Bus and the Configuration Management Process. It will keep trying this communication until it receives a reply from the configuration process. When the reply is received, the COMM process then asks for the network status. The configuration process then begins to send out the network status to all active COMM processes.

The COMM process is actually receiving the Network Status Table from the configuration process. Throughout the life of the network, the two processes periodically check each other to assure the status tables are identical and that the configuration process, and all COMM processes, are running properly. This is part of the backup system which is constantly checking itself for correct operation. The COMM process now has the status of all processes in the network as well as their respective NPU locations.

The COMM process could be implemented to ask the Configuration Management Process for the location of a global process each time it sends a message, but this would double the network message traffic. This would, obviously, tie up the NCB much more than necessary, and would lead to much slower response times. In addition, the duplication of the Network Status Tables in each NPU is an excellent way to provide backup for this most important single network function of configuration control.

The COMM process can determine where the receiving process is located from its status table and prepare the message for transmission to the proper remote NPU. The COMM process handles all formatting for transmission and all error checking at the receiving end of the transmission. During all inter-NPU messages, two identical COMM processes are talking to each other. An excellent protocol for this transmission is given in (Ref 7). This is the ARPANET's suggested protocol discipline for real-time highspeed data transmission, and would seem to be applicable as the standard for the proposed network.

The protocol suggested consists of the sending process noting the time that a message is transmitted and storing this time with the message in the process output queue. If after a preset interval the sending process receives no acknowledgement that this message was received correctly, the sending process will attempt another transmission of the message. This protocol precludes the use of negative replies to indicate a message was received in error. If the receiving process receives a message with an error, the process simply discards the message, knowing that it will be sent again. This helps to reduce message traffic by eliminating the negative acknowledgements and also provides detection of a failing piece of hardware when no acknowledgement has been received for a preset number of retrys. The entire protocol discipline is outlined in detail in (Ref 7) so no further details are included in this discussion.

The COMM process will include the handlers for the NCB hardware and would be responsible for handling both a serial and parallel protocol if both type of bus were included in the network. All functions of accomplishing the inter-NPU communication will be handled by the COMM process, thereby isolating this function from the rest of the network.

Intercom Terminal Management

One of the most important functions of the network is to provide better software preparation facilities for the users of the Digital Engineering Laboratory minicomputers. The network will provide better facilities than the minis alone, due to its access to all the network's IO devices, but this alone is not enough. It was mentioned earlier that the CDC 6600 does have the needed facilities available through its INTERCOM terminal system. The network terminals need a way to have access to the INTERCOM system and the INTERCOM Terminal Management process provides this access.

Before talking about the process functions, a word about the hardware interface to the CDC 6600. The network must match the protocol of its NCB to the protocol provided by the INTERCOM system. This appears to be another excellent application for a microprocessor, which could do the conversion necessary to translate the INTERCOM messages into standard network format messages. Since this conversion should not use the total capacity of the micro, a concentration function might also be added.

Fig. 9 shows how the micro would appear logically to the CDC 6600. Since the messages from all the network terminals would be going through the micro interface, the micro would appear to be multiplexing terminals A through E into the intercom system. Each network terminal would have its own logical connection to the INTERCOM system, but all messages to and from the terminals would be concentrated and multiplexed through one network link, the micro, to the CDC 6600.

While Fig. 9 shows the logical appearance of the network to the INTERCOM interface, Fig. 10 shows the actual message and control paths needed to accomplish this inter-

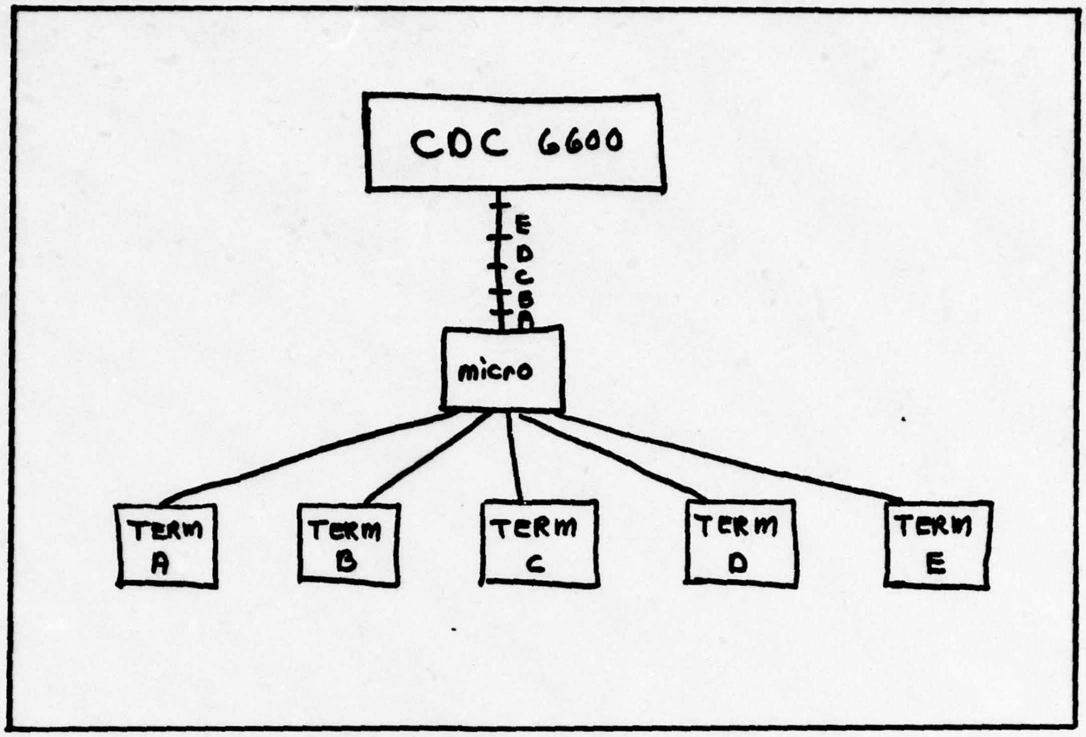


Fig. 9. The logical INTERCOM Interface

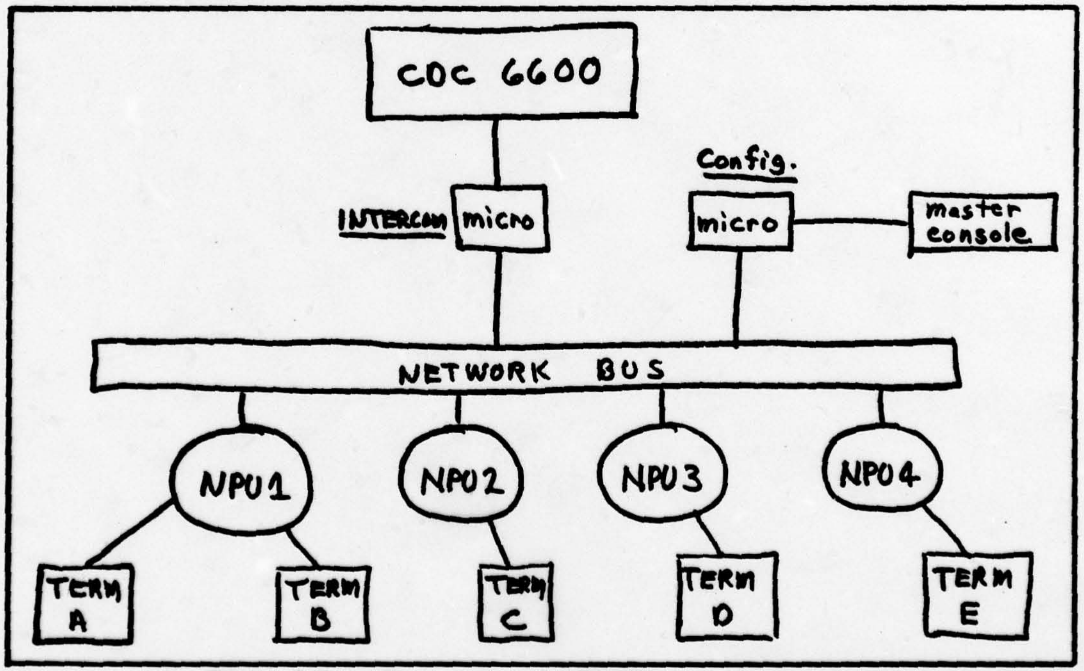


Fig. 10. The actual INTERCOM interface

connection. When the network is started up, the Network Status Table will indicate which of the network terminals are to be connected into the INTERCOM interface. This configuration can be changed at any time by the network operator. Therefore, the configuration process controls which network terminals will be allowed into INTERCOM at any time.

Once running, the INTERCOM Terminal Manager Process controls which of the terminals are actually logged on to the INTERCOM system. This process will output a message to each terminal assigned to it by configuration control indicating that the terminal is available for intercom use. It should be noted that no additional processes need to be added to any of the NPU's where the terminals are located, for all INTERCOM functions are handled by the INTERCOM manager in conjunction with the COMM processes and IO device processes through the standard network message protocol.

Performance Monitor

This network is to be an experimental tool to try out new and different methods of accomplishing many software and hardware functions. It must allow the students and faculty to compare different ways of handling the same function and to decide which method was better depending upon such factors as memory size, speed of execution, and reliability. Also, the network should be constantly improved to provide better service to all those using it.

Both requirements call for the necessity of the network to be able to monitor the performance of the network itself and the user jobs running on it.

Before the performance can be monitored, there must be data produced to monitor. This means that all of the processes described in Chapters V and VI must produce the data needed to allow their efficient monitoring. The key to the monitoring is again provided by the network message system.

The designer of each process must be aware of these statistics which might be of consequence to the performance monitoring effort and make provisions to output them as messages directed to a monitoring process. The User Job Scheduling Process is a good example. It would be almost impossible for a separate process to find out how much memory or execution time a user process is using if the statistics are not specifically made available.

Each time the User Job Scheduler starts a process, a message should be sent to the performance monitor containing memory usage and start time. Another message should be sent when the user process stops execution. In addition, the KERNEL will keep execution time on the NPU and output a message when the process is halted. All processes can be designed in this manner, using network messages to provide the necessary statistics to the performance monitor.

When all the proper information is provided by the system processes through system messages, the performance monitor has a much simpler job. It really becomes a passive process relying on the input messages to give a picture of now the network is running. Since this is a separate process, it too might be implemented in a microprocessor. By residing in a microprocessor, separate from all other processes, it is not using the execution time or resources of any of the NPU's which it is monitoring. This allows the network to be monitor with smaller effect than if the monitor was integral to one or more of the network supervisory functions.

The monitoring NPU (hopefully the micro shown above) should also have the capability of intercepting all network messages while they also go to the directed NPU. This monitoring NPU should be a special interface capable of tapping into all message traffic. In this way it can see not only the specific statistics directed to it, but it can also determine which NPU's are most active or which IO devices are receiving the most traffic. With the double capability of receiving specifically directed statistic messages, and being able to monitor all network message traffic, this performance monitor should provide an accurate picture of the inner network workings.

Data Base Management

Today many computer users are moving away from having individual data files for each specific application and are coordinating their files into large data bases. While this network will have many independent users, with many different needs in the data file area, many of the concepts used in data base management will apply. The most applicable concept is a standard access mechanism to all data files. This is the main concern of having a Data Base Manager Process.

This thesis has stated throughout that the user must be separated from having to know where resources are located in the network to use them. This is the purpose of the data base manager, to allow symbolic access to all data files in the network without regard to their location. It will also provide for the security access to the files.

Whenever a process needs access to a particular data file, it will send a message to the data base manager requesting a type of access (private or shared, read only, or read-write, etc.); the manager will check the access type requested by the process against its authorization tables and either grant or deny that access. If the request is granted, the data base manager will return the name of the process which handles the requested resource.

Working in conjunction with the Central Data Base Manager (CDBM) will be local resource managers (Fig. 11) which handle only the resources on one particular NPU. While all requests for initial access to a resource go to the central data base manager, once granted access the user process communicates only with the particular local

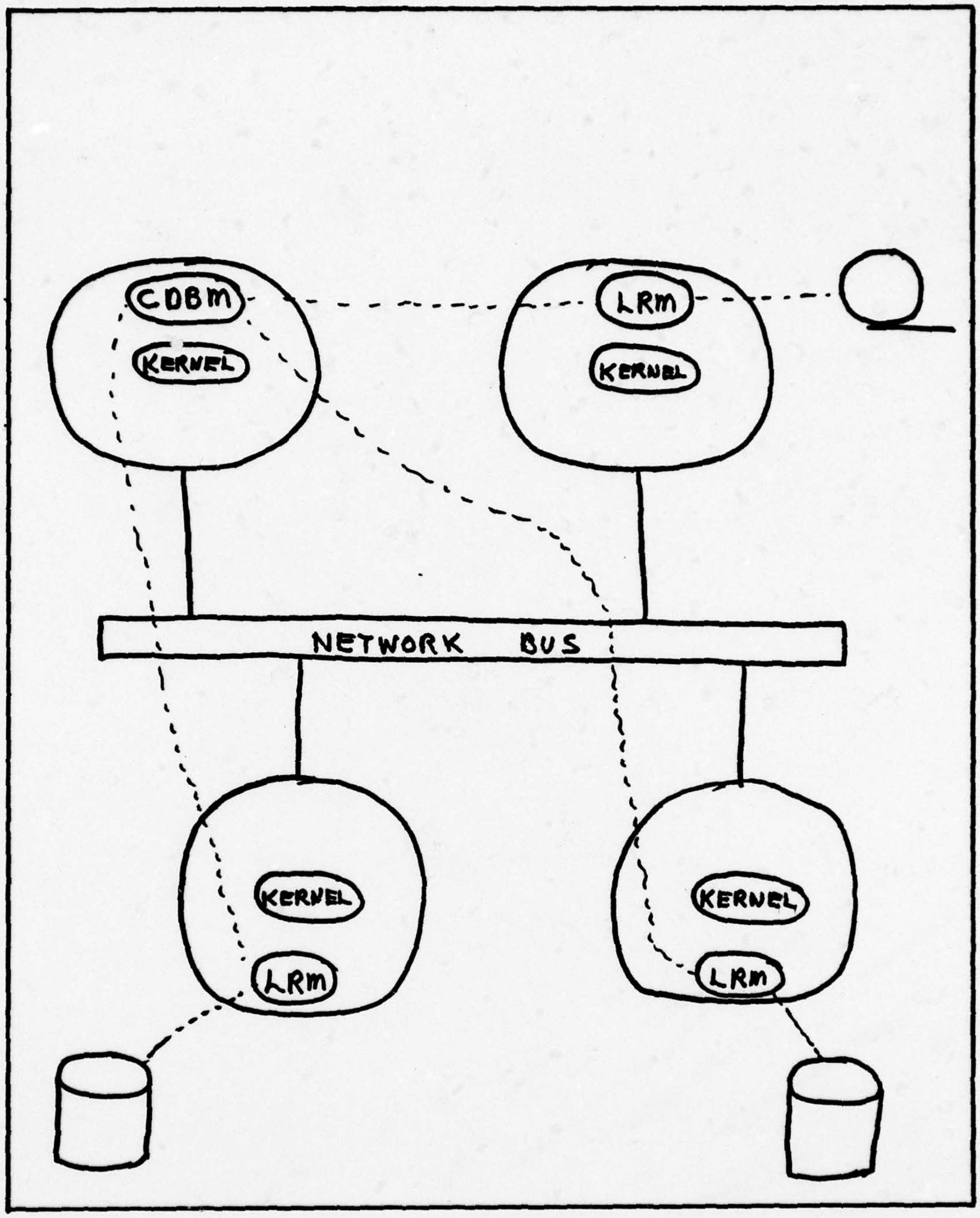


Fig. 11. The Data Base Manager

resource managing process. In this way a user process requesting access to a resource local to its NPU does not communicate across the NCB for all accesses - only for initial authorization access, which must be granted by the central data base manager. All normal data access (read and write) is handled by the Local Resource Manager (LRM) (Fig. 11) without causing unnecessary network bus traffic.

Once the user process has been given the name of its requested file's local resource manager, the user process proceeds as if that process is in total control. All further access to the data is done directly with the local resource manager. It can be seen that while the control of the network files is centrally located in one global data base manager, many processes are simultaneously handling the local data requests. This central control, with local access, provides an efficient method of handling all network data needs.

VII. CONCLUSIONS AND RECOMMENDATIONS

Introduction

This chapter discusses some conclusions reached from the development of a general architecture for a distributed mini-computer network. It also gives recommendations for a proposed plan for the detailed implementation of the network by follow-on thesis efforts.

Conclusion

Two things have been accomplished by this investigation. First, the functional requirements were developed to define what must be done to provide the AFIT faculty and students with a responsive mini-computer network in the Digital Engineering Laboratory. Next, an architecture was proposed which would lead to an implementation of hardware and software which would meet those requirements. The main emphasis of this thesis was given to the software architecture, although a basic hardware architecture was also presented.

The significant difference between the proposed network structure and the structures of most existing networks is the distributed independence of the network services. There is no central supervisor in this network. Each specific supervisory function is separated into an independent system process with all processes running in apparent concurrency.

The heart of the network is the independence of its processes. The processes have no common memory communication schemes, but instead communicate through a system of synchronizing messages. These messages allow each process to control its own execution synchronization and allow processes to communicate without regard to physical location in the network.

Any function can be changed in its entirety without forcing change in any other part of the network, due to the total separation of function into these independent processes. This independence enables the network to be truly distributed and divisible.

Recommendations

This thesis was intended to guide in the development of a Digital Engineering Laboratory mini-computer network. Since this was to be a guiding document, there was a temptation to suggest specific ways to produce the proposed architecture and delve into the many implementation details. While a few suggestions were given in the text, the author feels that further suggestions would only interfere with the implementers' own creative ideas and not necessarily produce a better product.

One bit of advice is offered. The order of presentation of the processes presented in Chapters V and VI might serve as an implementation schedule. The author chose this order for this particular reason. These processes seem to build on each other to provide a logical progression from basic needs through the "nice to have" but not entirely necessary functions. This is not to say that a data base manager is unnecessary, only that it may not serve the network to full capacity until the other functions preceding it are complete.

Bibliography

1. Baker, Bob E. Efficient Multiple Processor Coordination. Unpublished dissertation. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, November 1976.

2. Bohnsack, R. H. Interfacing HP21XX Minicomputers with a SEL 86 for Real-Time Applications. Unpublished thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, June 1976.

3. Clark, H. R., et al. Preliminary AFBITS Network Control System. ESD-TR-75-66. Springfield, Virginia: National Technical Information Service, 1975. ADA012218.

4. Coffin, D. D. and H.E.T. Connell. The Multi-Minicomputer Processor. ESD-TR-75-351. Washington: Defense Documentation Center, 1976. ADA021777.

5. Davies, E. W. and D.L.A. Barber. Communication Networks for Computers. New York: John Wiley and Sons, 1973.

6. Madnick, S. E. and J. J. Donovan.. Operating Systems. New York: McGraw-Hill Book Co., 1974.

7. Meyer, E. W., Jr. The RTDCOM Protocol - A Real-Time Intercomputer Data Transmission Protocol for the ARPA Network. Teledyne Geotech Technical Report. Springfield, Virginia: National Technical Information Service, 1975. ADA021687.

8. Mills, D. L. An Overview of the Distributed Computer Network. Computer Science Technical Report TR-413. College Park, Maryland: University of Maryland, 1975. ADA018734.

9. Warren, Hoyt M. A General Computer Network Simulation Model. Unpublished thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, March 1977.

Vita

Captain R. Cade Adams was born June 21, 1949 in Shreveport, Louisiana. After graduation from Byrd High School, Shreveport, Louisiana, he enrolled at Louisiana Polytechnic University at Ruston, Louisiana. Captain Adams received a Bachelor of Science degree in Computer Science in 1971 from Louisiana Polytechnic University and was commissioned in the United States Air Force through the Reserve Officer Training Corps immediately upon graduation. He was assigned to the 1155th Technical Operations Squadron at McClelland Air Force Base, California from the date of his commissioning until his assignment at Wright-Patterson AFB in June, 1976. He resides with his wife in Dayton, Ohio.

Permanent Address: 918 Delmar Avenue
Shreveport, Louisiana 71106

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/77-4	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) A DISTRIBUTED MINI-COMPUTER NETWORK		5. TYPE OF REPORT & PERIOD COVERED AFIT Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(*s*) R. Cade Adams, Capt, USAF		8. CONTRACT OR GRANT NUMBER(*s*)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) WPAFB OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December 1977
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*)		15. SECURITY CLASS. (*of this report*) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (*of this Report*)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (*of the abstract entered in Block 20, if different from Report*)

18. SUPPLEMENTARY NOTES

Approved for public release;
IAW AFR 190-17
Jerral F. Guess
Jerral F. Guess, Captain, USAF
Director of Information

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)

mini computer network
mini computer
distributed computer network

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)

This thesis presents a set of functional requirements for a distributed mini-computer network along with a basic architecture design to fulfill those requirements. The functional requirements were established by comparing the requirements of existing networks to the needs of the Air Force Institute of Technology Digital Engineering Laboratory. The requirements must provide a tool for education in operating systems studies while allowing mini-computer-based research.

cont. The primary emphasis of the proposed architecture is the separation of the network operating system functions into independent processes which will run concurrently on the various mini-computers in the network. This distribution of network supervisory functions is made possible by limiting all communication between processes to a set of snychronizing messages. The messages provide a data interface which allows communication without knowledge of the processes location within the network.

This investigation also discusses a set of processes which will perform the functions set forth in the requirements. A brief explanation of the functional working of each necessary process is provided.